Cloud-Based Acoustic Beamforming Emulator for FPGA-Based Sound Source Localization

Laurent Segers*, Bruno da Silva, An Braeken, Abdellah Touhafi
Department of Industrial Sciences and Technology (INDI), Vrije Universiteit Brussel,Brussels, Belgium
{laurent.segers, bruno.da.silva, an.braeken, abdellah.touhafi}@vub.be

***Abstract*—Acoustic beamforming techniques are often used in applications involving microphone arrays such as sonar, binaural hearing aid devices and acoustic indoor localisation. The beamforming methods utilized describe results for a particular array within a given acoustic frequency domain. Software and toolboxes running locally facilitating the theoretical design of such implementations are largely available. However, a multi-user cloud-based beamforming approach to generate results could not be found. Also, generating packages facilitating the implementation on FPGAs has not been carried out to our knowledge. In order to alleviate these shortcomings, we propose a scalable cloud-based accessible multi-user beamforming emulator where users launch emulations based on acoustic beamforming for microphone arrays. Users create, upload, modify and run beamforming emulations. Our platform is also able to generate VHDL packages containing delaying tables which facilitates the implementation of HDL based Delay-and-Sum beamformers. Analysis based on algorithm truncation errors are also possible due to the integration of fixed-point signal processing algorithms. Microphones in a given array can be disabled in order to estimate the response of the remaining microphones during power saving. Results and packages can be downloaded from the user web interface as an archive file. Our Emulator can be configured to take several user defined configurations into account such as microphone arrays, sound source placement and emulation parameters. Graphs such as waterfall diagrams and directivity showing the quality of the beamforming can also be generated.**

***Index Terms*—Cloud-Based Acoustic Beamforming, Microphone Array Beamforming, FPGA Microphone Array Beamforming Emulator, Delay-and-Sum Cloud-Based Emulator**

I. Introduction

In recent years, advances in Micro Electro-Mechanical Systems (MEMS) microphone technology and acoustic beamforming techniques allow for enhanced sound source localization in both acoustic and ultrasound frequencies [2], [3], [8]. Sound source localization based on microphone arrays have emerged and are used in various applications; ranging from ultrasound source localization [1], speech localization [7], binaural hearing aid for disabled people [4] and sonar [5]. Advances in embedded platform technologies allow the possibility to implement beamforming algorithms for microphone arrays on reconfigurable architectures such as Field Programmable Gate Arrays (FPGAs) [8]–[11]. The placement of the microphones, the utilized algorithms and the amount of microphones determine the level of the beamforming accuracy in terms of acoustic frequency resolution and spatial resolution. Estimating the response of a microphone array based prior to a hardware implementation has been achieved by da Silva et al. [9]. There we emulated the FPGA algorithms on a local computer in order to obtain optimized algorithms before the implementation on the hardware. Aside of calculating the directivity of the array in one direction, the 95 % confidence interval of the directivity when steering in 360° has been calculated. Matlab offers the possibility to calculate the response of a microphone array through the additional "Phased Array System Toolbox" [13]. J. Steckel et al. used ultrasound beamforming to develop and improve a 3D enabled sonar which allows to extract information of the environment using sparse arrays of microphones in conjunction with a single Polaroid emitter [1]. Sun et al. [6] propose an improved direction of arrival (DoA) technique based on the generalized cross correlation algorithm in conjuction with a probabilistic neural network approach for enhanced sound source localization in noisy and reverberant rooms. Most beamforming tools allow to perform an analysis of a given microphone array along with the selected beamforming algorithm. In WaveCloud [14], [15] an open source simulation tool for acoustic sound propagation in buildings is proposed. Users are free to download and install the tool. WaveCloud imports a 3D-stl file of the building and users can place sound sources in the simulation environment. All the previously discussed tools are executed locally and most are computational intensive and prohibits the use of the local machine for other tasks. Moreover, these tools are bound to a group of users having access to that particular machine.

In a recent movement, several companies offer cloud-based simulation alternatives to the well-established engineering tools. Simscale [17] offers an online simulation platform for Computational Fluid Dynamics (CFD), Finite Element Analysis (FEA) used in structural eningeering and thermal propagation in materials. This cloud-simulation tool is proposed in a free limited version as well as in paid versions for professionals. Waveller Cloud from KUAVA [16] is a simulation tool which allows to simulate acoustic propagation in prototypes such as the engines of cars, gearboxes, etc. They provide their tool as a "Software as a Service" (SaaS) and users pay on a monthly basis or per CPU-hour cost. Aside of other cloud-based simulation tools, none of the tools provide the ability to simulate the beamforming of a user defined microphone array and to facilitate the implementation of FPGA implementations by means of on the fly generated VHDL packages. In order to allow other researchers to grant access to our emulation without exposing the code and the associated learning curve, we propose a cloud-based platform

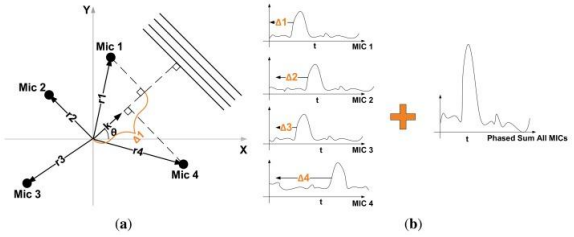

Fig. 1. Principle of the Delay-and-Sum beamforming algorithm. Samples are first delayed by a given amount of time before added [8].

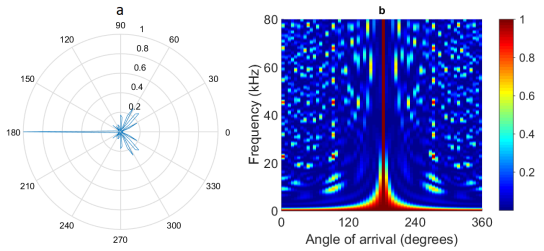

Fig. 2. Polarplot depicting the direction of arrival of a 30 kHz wave at 180° (**a**). Here 64 steering orientations have been used to calculate the P-SRP. Aside of the main lobe pointing in the direction of the sound source, several grating lobes are also visible due to the spatial aliasing caused by the microphones' placement. A waterfall diagram (**b**) depicting the frequency response of an array between 1 kHz and 80 kHz.

where users can generate, modify and download emulations. The calculation of the beamforming and the generation of the VHDL packages are offered as a SaaS to facilitate local implementation onto FPGA boards. The remaining of this paper is structured as follows: in Section II we describe the background in acoustic beamforming for microphone arrays. In Section III we demonstrate the architectural overview of our implementation, followed by the description of the beamforming Emulator. Then we describe the multi-user web interface along with the corresponding database structure. We also describe the back-end link between the online web interface and the Emulator. In Section IV we present the online environment and the obtained output results which can be downloaded from the platform. In Section V we conclude our and propose future work implementations.

II. Acoustic Beamforming

Acoustic beamforming methods utilize the acoustic signals of each of the microphones in an array such that the bearing of a sound source can be found. The main objective of a beamforming algorithm is to amplify the acoustic signals coming from a specific direction while suppressing the signals coming from all other directions. This is achieved by delaying the samples of the microphones in the array by a given amount of time such that the combination of these signals results in a signal amplification. The steered response power (SRP) is than obtained by calculating the signal power at that direction. When applying this principle to all desired steering directions, one can obtain a polar steered response power (P-SRP) describing the intensity of the incoming sound waves at particular directions of arrival. The normalized P-SRP can be represented by means of a polar plot (see Figure 2). A well known beamforming algorithm is the Delay-and-Sum method, where the samples are added after proper delaying. The principle of Delay-and-Sum is given in Figure 1. Variants on the regular Delay-and-Sum have been carried out by da Silva et al. [9], [10].

A complete frequency response of the microphone array can be calculated by applying monochromatic sine waves of increasing frequencies against the microphone array. All the obtained P-SRPs can be plotted in a waterfall diagram. An example of a waterfall diagram is given in Figure 2. The algorithms described here calculates the response for a planar 2-dimensional environment [8]. However, the principles remain similar in a 3-dimensional setup [12].

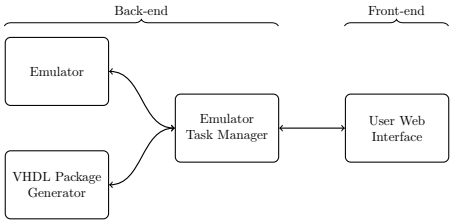

Fig. 3. Architectural overview of the Emulator chain. On the left side the back-end applications can be retrieved while the front-end application can be found on the right side. The back-end application can be ran on several computers for concurrent calculations.

III. Architectural Overview

Our cloud-based emulator consists of 4 major parts: the User Web Interface, the Emulator and the VHDL Package Generator and finally the Task Manager. A general overview is shown in Figure 3.

The Emulator calculates the acoustic beamforming response for a given array and sound source pattern. The VHDL Package Generator is responsible for generating a VHDL package containing the necessary information with the delaying tables which facilitates the implementation for beamformer based algorithms on FPGAs. The User Web Interface allows several users to upload, create or modify emulations which can be processed by the Emulator. The web interface also contains a database where all emulations can be stored and queued for later emulations. The Task Manager allows to launch the Emulator with the requested operations queued by the web interface in an asynchronous way and stores the results to the appropriate user. The Task Manager also issues commands to the VHDL Package Generator for VHDL package generation. The complete system runs on a Ubuntu Linux 16.04 LTS server.

A. Emulator

The Emulator computes the frequency responses for a user defined microphone array along with a given set of processing

parameters. The processing flow of the Emulator is defined in 3 phases.

Our Emulator is written in Matlab and runs as a standalone application on any Matlab enabled machine.

1) Phase 1: Processing input parameters: The Emulator computes the required frequency responses for a given set of parameters and for a given shape of microphone array. The necessary configurations are stored into configuration files which are processed by the Emulator. This approach offers the advantage that any shape of microphone arrays can be defined by users. For each emulation to be computed, 3 configuration files are required:

- emulation file: file for the emulation parameters,
- array file: a file containing the microphone array properties and,
- sound source file: a file containing the sound sources around the microphone array.

Each category of configuration files is stored into a separate input folder. The file containing the emulation parameters refers to the appropriate sound source and array configuration files. Aside of this, the emulation parameters also contain the targeted acoustic frequency range, the type of microphone to be utilized, the required beamforming algorithm, the sampling and decimation frequencies and the required digital filtering chain. Since the Emulator emulates the behaviour of the beamforming algorithm on the FPGA, it also includes the necessary bit-width parameters which should be used during the calculations. At last, the parameters do also include the possibility to generate several output graphs for the metrics such as a waterfall diagram, polar plots at several frequencies and the directivity of the input array. The file containing sound sources holds the Cartesian position of each of the sound sources used in the emulation.Each source is also given a certain emitting amplitude and frequency, which can be fixed or variable. Variable frequencies are then given by the emulation file. Fixed frequencies can be used to emulate noise sources around the array. The array configuration file contains the parameters to define the Cartesian position of each of the microphones in the array. Each array configuration file also defines how many groups (i.e. sub-arrays) of microphones can be defined. Each sub-array can be deactivated allowing a user to compare the response of the array when all microphones are active, or when only a fraction of the array is turned on for power and resource savings. The activation or deactivation of the sub-arrays is defined in the emulation file.

2) Phase 2: Beamforming Frequency Response Computation: During this step, the frequency response of the microphone array is computed. The computation of the frequency response can be summarized by the following steps:

1) Placement of the sound sources according to the configuration file.
2) Sound sources generate acoustic waves radiating towards the microphone array.
3) Sampling of the acoustic waves at each of the microphone's positions. Depending on the microphone type, the sampled data is either in pulse code modulation (PCM) or in pulse density format (PDM).
4) Computation of the chosen beamforming method.
5) Computation of the output response and directivity.

During the first steps, the distances between the sound sources and the microphone is computed and leads to an acoustic propagation delay measured by each of the microphones.

The delayed acoustic waves are sampled by each of the microphones. Frequency characterization against the frequency response of the selected type of microphone is first applied in order to mimic the real behaviour. Once the incoming wave has been reshaped, the proper sampling occurs. This can be done in PCM or in PDM format. Sampling the acoustic waves in both PDM and PCM format offers the advantage to mimic most types of microphones at hardware level. The PDM signals are obtained through sigma-delta modulation (SDM) techniques such as found in digital PDM microphones. The calculation of the necessary parameters for the PDM modulators can be obtained by using the Delta Sigma (delsig) toolbox [18]. Once the sampling is accomplished, the Emulator feeds the samples at the requested beamforming algorithm. Several implementations are proposed such as delay-filter-sum and filter-delay-sum. In the former case, the samples are first delayed before being filtered and summed. In the latter beamformer algorithm, the samples are first filtered before being delayed and summed. At last, the directivity of the beamforming is computed, demonstrating the possibilities of a given microphone array. During the beamforming stage, one can either opt for theoretical results by requiring computations in double precision format or in fixed-point precision mimicking FPGAs where fixed-point computations require less resources.

3) Phase 3: Post processing: Once the beamforming has been computed, results are stored into an output project folder, containing the calculated results in a comma separated file (CSV-file), the 3 configuration files and the output graphs generated during post processing. Generating graphs can be enabled by setting the appropriate flags in the emulation configuration file. These flags include the possibility of generating waterfall diagrams, polar plots for each computed frequency and plots for the directivity metric.

The complete folder structure containing the input and output files is shown in Figure 4.

B. VHDL Package Generator

Aside of our Emulator, a VHDL Package Generator has been developed which facilitates the creation and actualization of delay-and-sum beamformers described in Hardware Description Languages (HDL). The selection of the VHDL language allows the portability of the HDL beamformer's description since it is independent of the FPGA's vendor. The package contains the calculated delays required to perform the delay-and-sum beamforming operation. The characteristics of the microphone array under evaluation are used to adjust the required on-chip memory blocks to properly delay the input signals. An advantage of the automated generation

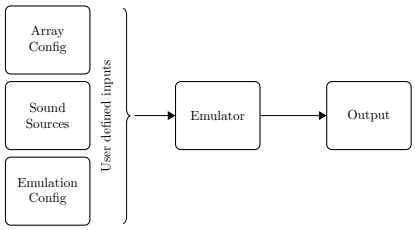

Fig. 4. Folder structure of the Emulator. On the left the 3 input folders containing the required configuration files of the user, on the right the output folder where the results of the Emulator are computed. An additonal configuration folder for the Emulator enables to port the Emulator onto other machines.

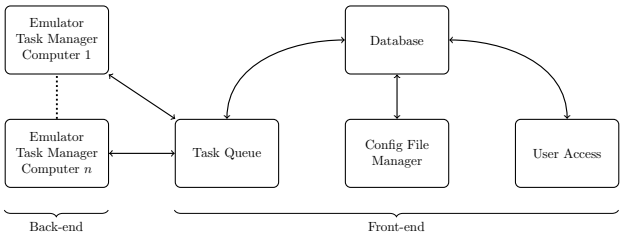

Fig. 5. User web interface. On the left the applications communicates with the Emulator Task Manager while user access and the configuration file manager are on he right side. Several computers (n computers) are allowed to work concurrently on different tasks.

of the VHDL package is the flexibility when defining the microphone array configuration. As one of the features of our Emulator, the microphones of the array can be grouped in sub-arrays. This is interesting when exploiting microphones' features such as *sleep mode* (i.e. such as the SPH0641LU4H-1 microphone [19]), which allow to disable microphones at runtime. The microphones drastically reduce their power consumption during *sleep mode*. The proper selection of the microphones composing a sub-array is not trivial, since the position of the microphones directly affect to the array's response. The flexibility to explore sub-arrays supported by our Emulator is exploited when defining in HDL language the required on-chip memory to perform the signal delays. The defined sub-arrays are treated independently when generating the on-chip memory block HDL description. The benefit is a significant memory reduction since that the maximum delay required by each sub-array relates to the consumed memory. The decomposition in sub-arrays, however, demands a certain adjustment when combining the signals coming from different sub-arrays. The automated VHDL package uses the maximum delay of each sub-array to determine when the output data from the beamformer is synchronized. Constant values related to the beamforming operation and the bit-width of the memory addresses and internal buses are also automatically determined when generating this VHDL package.

C. User Web Interface

Aside of the back-end Emulator, the User Web Interface plays an important role by managing the emulations provided by the users. The most important task for the interface is to provide an accessible user interface which is able to manage the requested operations. This includes safe editing of emulations, creating emulations and removing emulations but also enabling users to download and extract computed information in an accessible and private way. Therefore, the User Web Interface is subdivided into 3 major parts. The complete interface is written in php and utilizes a MySQL database for keeping track of users and user data. A general overview of the user web interface is shown in Figure 5.

1) User Management: Users access the system and utilize editing tools for running emulations or generating requested VHDL packages. Users are able to log in, log out. Each user has following characteristics stored in the database: a user ID, a username, name, given name, a password and an e-mail address. Users may also set a flag enabling (or disabling) sending reports to the user via e-mail. This is especially useful since emulations can take several hours to be computed.

Each enabled user starts with a private root folder space where all configurations files are stored. Each user root folder is further subdivided in the sub folders containing the emulation, sound source and array configuration files. The folder structure is equivalent to the folder structure of the Emulator. A folder for results is also added. The utilization of a separate root folder for each user ensures data integrity by removing the risk of a user editing the input files while the Emulator calculates the requested output on the same input files.

2) Configuration File Management: Each user is able to upload, to edit or to remove configuration files from his/her own defined root folder. Each configuration file is linked in the database. This approach also holds for the generated results. Each link in the database contains following information: A file ID: a unique ID for each file (primary key), the file name, the complete path to the file relative to the root folder of the user, the file type: sound source, array, emulation configuration or results file (zip) and the ID of the owning user.

The User Web Interface also performs the necessary validations of the input configurations in order to ensure a correct work flow of the back-end Emulator. Therefore, all available configuration parameters which are taken into account by the Emulator are stored in the database along with the required usability constraints. E.g. a waterfall diagram can only be computed if the frequency range spans at least over 2 different frequencies. A short list of parameters with the constrains is given below in table I. Configurations containing errors can not be launched and appropriate errors are shown to the user.

3) Task Queuing: Once a requested operation is ready to be processed by the Emulator, the user web interface queues the requested operation for later processing in a queuing table. A flag set to "to be processed" ensures that this operation will be processed by the back-end Emulator. The emulation task manager communicates with the database by means of php managing scripts to dequeue required tasks.

D. Emulation Task Manager

The last part of the architecture is the Emulation Task Manager. The Emulation Task Manager ensures the communication

TABLE I
EMULATION PARAMETERS ON WHICH A VALIDITY CHECK IS PERFORMED BEFORE BEING QUEUED FOR EMULATION.

Parameter	Constraints
Beamforming method	A method available in the Emulator
# steering angles	> 0
Decimation filter stage	A method available in the Emulator
Sampling frequency (f_s)	$0 < f_s < fs_{max_{microphone}}$
Delay table frequency	$\leqslant f_s$
Require VHDL package	True or False
Bitwidth of signal processing	0 (double precision), > 0 (fixed-point)
Array config file	Available array config file
Sound sources config file	Available sound source file
Emulation name	A user defined name
Start frequency (f_{start})	user defined, but $f_{start} \geqslant 0$
Stop frequency (f_{stop})	$\geqslant f_{start}$
Frequency increment (f_{inc})	> 0
Microphone type	Microphone type available the Emulator
Sound sampling method	PCM or PDM
Generate graphs	True or False
Generate waterfall diagram	if "Generate graphs"=True and $\geqslant 2$ frequency responses requested
Generate polar plot	if "Generate graphs"=True
Generate directivity	if "Generate graphs"=True and $\geqslant 2$ frequency responses requested
Generate MSL	if "Generate graphs"=True and $\geqslant 2$ frequency responses requested

between the web interface and the Emulator. This application is written in C++ and communicates with the web interface by means of the CURL REST-API. Launching the Emulator with the Matlab engine is done by issuing terminal commands.

First, the manager issues a POST request to the web interface for available queued tasks. Only tasks which are flagged as "to be processed" are issued by the manager. If a task is available, the web interface replies with the given task by means of a user identification packet together with the necessary input files to be processed by the Emulator. The files to be processed are also downloaded from the web environment to the Emulator. Depending on the specified operation in the input files the manager launches the Emulator or the VHDL package generator. Once the operation command on a given set of input files has been issued, the manager issues a POST request in order to flag the task as "being calculated". The Matlab engine returns after the requested computations and the manager collects the output by compressing the results to a zip archive which is uploaded to the account of the corresponding user. Once all steps are finished the manager issues a POST command to the web interface to flag the current task as being "completed". Along with this POST action the resulting output zip file is stored into the database for the later retrieval by the user. Once the loop of operations is completed, the managers issues a POST request in order to obtain a new task. If no task has been found available, the manager process idles and awakens every minute to re-check for available tasks to be issued. This web-based communication between web interface and the Emulation Task Manager allows to run the background emulations on several computers while users only see one web environment. Computers available for computations can issue new commands while other computers

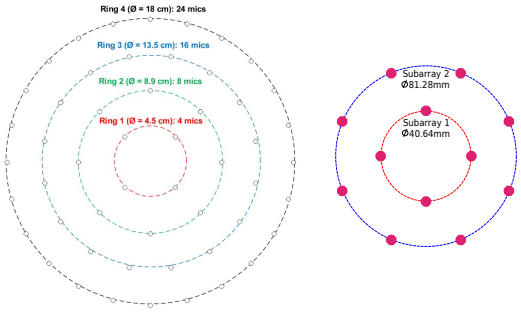

Fig. 6. The soundcompass array (Left) consisting of 52 microphones placed in 4 rings. Each ring corresponds to a sub-array which can be enabled or disabled and the smaller microphone array (Right) consisting of 12 microphones placed in 2 rings. Each ring corresponds to a sub-array which can be enabled or disabled.

are concurrently computing other requested operations. This scalability also enables to dynamically increase or decrease the available processing power on demand.

IV. DEMONSTRATION AND RESULTS

In this section we present the final application which is shown to the users. Users can log in the platform and create, modify and queue desired emulations or VHDL package generating tasks. In order to show the configurable possibilities of our Cloud-based Emulator platform, we emulate the behaviour of 2 different microphone arrays using 2 different types of microphones. The complete system runs on a computer sporting a Q6600 CPU with 8 GB of RAM.

The first array we will evaluate is the soundcompass microphone array from Tiete et al. [8] consisting of 52 ADMP521 microphones [20]. The distribution is shown in Figure 6.

The complete array is subdivided into 4 sub-arrays which correspond to each of the microphone rings. The second array to be evaluated is a smaller array consisting of 12 SPH0641LU4H-1 ultrasound microphones [19] (Figure 6).

Two sub-arrays corresponding to each ring of microphone can be defined. Due to the frequency limitation of the microphones of the first array, the response will be evaluated from 20 Hz up to 16 kHz in steps of 20 Hz. Since the second array consists of ultrasound enabled microphones, it will be evaluated from 100 Hz up to 80 kHz in steps of 100 Hz. In both cases the microphones utilize a PDM modulator to generate acoustic signals. We also request to generate complete waterfall diagrams for a sound source located at $180°$. The directivity plots are also shown.

The requested response of the complete Soundcompass array is computed and the results are downloaded after computation in an archive file (Figure 8). The archive contains the waterfall diagram and directivity plot (Figure 7). The calculations took approximately 5 hours to complete on the 4 available cores. Similar calculations are preformed on the complete smaller array and results are also retrieved. The figures containing the waterfall diagram and the directivity are shown in Figure 7. The calculations took approximately 1 hour and 15 minutes to complete. Computing a single P-SRP as shown in Figure 2 where only 1 frequency is required

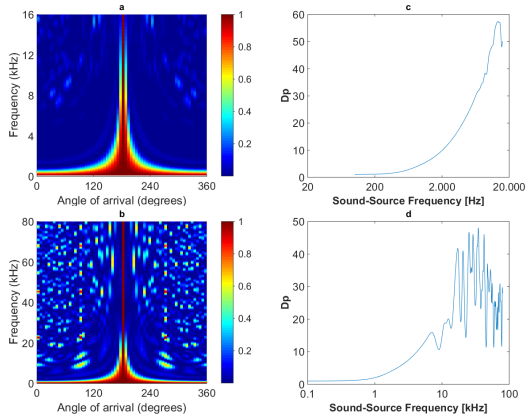

Fig. 7. Waterfall diagram (**a**) of the soundcompass and the waterfall diagram (**b**) of the smaller array when all microphones are enabled. The corresponding directivities D_p are shown in graphs **c** (soundcompass) and **d** (smaller array).

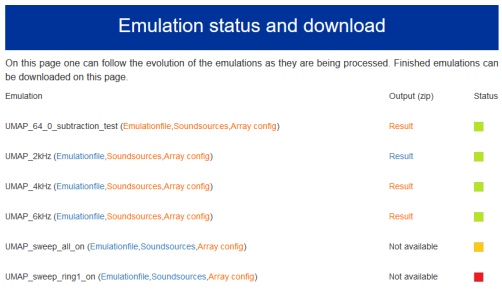

Fig. 8. The progress of the emulations while being computed shown in the "Emulation status and download" web page. Only the 4 first emulations can be downloaded, the last two are in progress (orange) or red (to be computed).

takes around 90 s. Here, the computations of a single frequency response is done on a single core due to Matlab limitations regarding memory allocation and thus comparatively requires more time than running a full frequency analysis.

The proposed environment has been designed to be able to operate on a multi-computer environment. Evaluation on a multi-computer towards scalability is still ongoing research.

V. Conclusions and Future Work

In this paper we present our Cloud-based Emulator for acoustic beamforming applications and for FPGA ready implementations. The current platform is able to compute requested operations and to generate HDL-ready implementations for a given microphone array and emulation parameters. Our platform is also able to generate diagrams for the directivity and waterfall diagrams. Several types of microphones can be requested, sampling in PCM and PDM format. Users are able to define microphone sub-arrays in order to compare results when enabling or disabling groups of microphones. Our emulator can also take advantage of a multi-computer environment in order to distribute the computing tasks.

VHDL packages containing delays can be requested from our platform. However, we envisage to add the possibility to generate a VHDL package containing the complete beamforming logic for the FPGA. By doing so, the user would only be required to have the FPGA with the corresponding tools installed on a local computer. This would eliminate the VHDL development cost and significantly reduces the time required when targeting a new microphone array.

References

[1] J. Steckel, "Sonar System Combining an Emitter Array With a Sparse Receiver Array for Air-Coupled Applications", in IEEE Sensors Journal, vol. 15, no. 6, pp. 3446-3452, June 2015. doi: 10.1109/JSEN.2015.2391290

[2] Segers, L.; Van Bavegem, D.; De Winne, S.; Braeken, A.; Touhafi, A.; Steenhaut, K. "An Ultrasonic Multiple-Access Ranging Core Based on Frequency Shift Keying Towards Indoor Localization". Sensors 2015, 15, 18641-18665

[3] Segers, L.; Tiete, J.; Braeken, A.; Touhafi, A. "Ultrasonic Multiple-Access Ranging System Using Spread Spectrum and MEMS Technology for Indoor Localization". Sensors 2014, 14, 3172-3187

[4] E. Hadad et al., "Comparison of two binaural beamforming approaches for hearing aids," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, 2017, pp. 236-240. doi: 10.1109/ICASSP.2017.7952153

[5] J. Steckel, A. Boen and H. Peremans, "Broadband 3-D Sonar System Using a Sparse Array for Indoor Navigation," in IEEE Transactions on Robotics, vol. 29, no. 1, pp. 161-171, Feb. 2013. doi: 10.1109/TRO.2012.2221313

[6] Y. Sun, J. Chen, C. Yuen, S. Rahardja, "Indoor Sound Source Localization with Probabilistic Neural Network", IEEE TIE, vol. 65, Aug 2017, pp. 6403-6413.

[7] Brandstein, Michael S. and Harvey F. Silverman. "A practical methodology for speech source localization with microphone arrays." Computer Speech and Language 11, 1997, 91-126.

[8] Tiete, J.; Dominguez, F.; Silva, B.D.; Segers, L.; Steenhaut, K.; Touhafi, A. SoundCompass: A Distributed MEMS Microphone Array-Based Sensor for Sound Source Localization. Sensors 2014, 14, 1918-1949.

[9] da Silva, Bruno, et al. "A Low-Power FPGA-Based Architecture for Microphone Arrays in Wireless Sensor Networks." International Symposium on Applied Reconfigurable Computing. ARC 2018. Springer International Publishing, 2018.

[10] da Silva, Bruno, et al, "Design Considerations When Accelerating an FPGA-Based Digital Microphone Array for Sound-Source Localization," Journal of Sensors, vol. 2017, Article ID 6782176, 20 pages, 2017. https://doi.org/10.1155/2017/6782176

[11] da Silva, Bruno, et al. "A Multimode SoC FPGA-Based Acoustic Camera for Wireless Sensor Networks." 2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). IEEE, 2018.

[12] Taghizadeh, M.; Garner, P.; Bourlard, H. Microphone Array Beampattern Characterization for Hands-Free Speech Applications. Proceedings of the IEEE 7th Sensor Array and Multichannel Signal Processing Workshop, Hoboken, NJ, USA, 1720 June 2012; pp. 465468.

[13] MathWorks, "Phased Array System Toolbox", Online: https://nl.mathworks.com/products/phased-array.html

[14] Sheaffer, J and Fazenda, BM, "WaveCloud : an open source room acoustics simulator using the finite difference time domain method", 2014, Forum Acusticum

[15] Jonathan Sheaffer, "WaveCloud-M: Acoustics FDTD simulator for Matlab", Online: http://www.ee.bgu.ac.il/ sheaffer/wavecloud.html

[16] KUAVA, "Waveler Cloud", Online: http://www.kuava.fi/software-solutions/waveller-audio-and-noise-simulation-system/wavecloud.html

[17] SimScale, Online: https://www.simscale.com/

[18] Richard Schreier, "Delta Sigma Toolbox", Online: https://nl.mathworks.com/matlabcentral/fileexchange/19-delta-sigma-toolbox

[19] SPH0641LU4H-1 microphone datasheet, Online: https://www.mouser.be/datasheet/2/218/-746191.pdf

[20] ADMP521 microphone datasheet, Online: http://www.analog.com/media/en/technical-documentation/obsolete-data-sheets/ADMP521.pdf